A Hierarchical Resource Reservation Algorithm for Network Enabled Servers

Eddy Caron, Frédéric Desprez, Franck Petit, Vincent Villain

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Hierarchical Resource Reservation Algorithm for Network Enabled Servers

E. Caron , F. Desprez , F. Petit , V. Villain

No 4701

January 2003

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--4701--FR+ENG

A Hierarchical Resource Reservation Algorithm for Network Enabled Servers

E. Caron , F. Desprez , F. Petit , V. Villain

Thème 1 — Réseaux et systèmes
Projet ReMaP

Rapport de recherche n ° 4701 — January 2003 — 13 pages

Abstract: This paper presents the application of the PIF algorithm to a Network Enabled Server environment. Hierarchical scheduling is applied to improve the scalability of the overall architecture and fault tolerance problems are addressed using timers. The simulation shows that gains can be obtained using such a platform over single scheduler approaches.

Key-words: Distributed computing, hierarchical scheduling, Network Enabled Servers.

(Résumé : tsvp)

This text is also available as a research report of the Laboratoire de l'Informatique du Parallélisme `http://www.ens-lyon.fr/LIP`.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN (France)
Téléphone : 04 76 61 52 00 - International : +33 4 76 61 52 00
Télécopie : 04 76 61 52 52 - International : +33 4 76 61 52 52

Algorithme hiérarchique de réservation de ressources pour des serveurs de calcul distribués

Résumé : Cet article présente l'algorithme du PIF dans un cadre de serveurs de calcul distribués. L'extensibilité de la plate-forme ainsi que des mécanismes de tolérance aux pannes sont mis en œuvre via un ordonnancement hiérarchique. Le gain de performances est mis en évidence par un outil de simulation.

Mots-clé : Calcul distribué, Ordonnancement hiérachique, Serveurs de calcul

1 Introduction

Huge problems can now be computed over the Internet with the help Grid Computing Environments [10]. Several approaches co-exist like object-oriented languages, message passing environments, infrastructure toolkits, Web-based, and global computing environments, ... The RPC paradigm seems also to be a good candidate to build Problem Solving Environments (PSE) for different applications on the Grid. Several tools following this last approach exist, like NetSolve [2] or Ninf [12]. They are commonly called Network Enabled Server (NES) environments [11] and usually have five different components: *Clients* that submit problems they have to solve to*Servers*, a *Database* that contains information about software and hardware resources, a *Scheduler* that chooses an appropriate server depending on the problem sent and the information contained in the database, and finally *Monitors* that acquire information about the status of the computational resources.

The environments previously cited have a centralized scheduler which can become a bottleneck when many clients try to access several servers. Moreover as networks are highly hierarchical, the location of the scheduler has a great impact on the performance of the overall architecture. Thus we have designed DIET [4], a NES environment that focuses on offering such a service at a very large scale using a hierarchical set of schedulers. The scheduling algorithm has to take into account the hierarchy and the possibility of simultaneous request filling the different stages of the tree. Moreover, fault can occur at the server or at the scheduler level.

Few work exist about scheduling in NES environments. In [16], the authors present an algorithm that uses dead-lines on requests to load balance the work among the servers. However, the scalability of the overall architecture is not studied. The authors of [1] present H-SWEB, a hierarchical approach for the scheduling of HTTP requests across clusters of servers. Hierarchical scheduling can also be applied to shared memory machines like in [7] or distributed memory machines [17] for the scheduling of independent jobs. Finally, in [14], the authors present a 2 level scheduler for metacomputing systems.

Our paper presents the extension of an existing algorithm (PIF) used for resource reservation in a NES environment using a hierarchy of schedulers. This algorithm can take into account faults at different levels using timers.

The rest of the paper is organized as follows. The first section present the DIET environment and its hierarchical architecture. Section 3 describes the PIF algorithm and the extensions we added for fault tolerance. Finally and before some concluding remarks, we present a validation of the algorithm and the hierachical approach using simulations.

2 DIET Overview

DIET [4] is built upon *Computational Resource Daemons* and *Server Daemons*. The scheduler is scattered across a hierarchy of *Agents*. Figure 1 shows the hierarchical organization of DIET. A **Client** is an application that uses DIET to solve problems. Different kinds of clients should be able to connect to DIET from a web page, a PSE such as Matlab or Scilab, or from a program written in C or Fortran. Computations are done by servers in front of which we have **Server Daemons (SeD)**. A SeD encapsulates a computational server. For instance it can be located on the entry point of a parallel computer. The information stored on an SeD is a list of the data available on its server (with their distribution and the way to access them), the list of problems that can be solved on it, and all information concerning its load (memory available, number of resources available, ...). A SeD declares the problems it can solve to its parent LA. A SeD can give performance prediction for a given problem using the performance evaluation module (FAST). The hierarchy of scheduling agents is made of a **Master Agent (MA)**, several **Agents (A)**, and **Local Agents (LA)**.

A Master Agent receives computation requests from clients. These requests refer to some DIET problems listed on a reference web page. Then the MA collects computation abilities from the servers and chooses the best one according to some scheduling heuristics. A reference to the server chosen is sent back to the client. A client can be connected to an MA by a specific name server or a web page which stores the various MA locations. An Agent aims at transmitting requests and information between MAs and LAs. The information stored on an Agent is the list of requests and the number of servers that can solve a given problem and information about the data distributed in this subtree. Depending on the underlying network topology, a hierarchy of Agents may be deployed between an MA and the LAs. Finally, a Local Agent aims at transmitting requests and information between Agents and several servers.

DIET includes a module called FAST(*Fast Agent's System Timer*) [13] to provide different information needed by the agents. FAST is a software package allowing client applications to get an accurate forecast of routines needs in terms of completion time, memory space and communication costs, as well as of current system availability (memory, machine load), and communication speeds. For sequential routines, we developed a tool [9] that benchmarks routines in time and space and then fits the resulting data by polynomial regression. Concerning parallel versions, analytical expressions were extracted from code studies to give a theoretical model [5]. The forecast the communication time between different elements of the hierarchy is allowed by the use of NWS (Network Weather Service) [18]. NWS sensors are placed

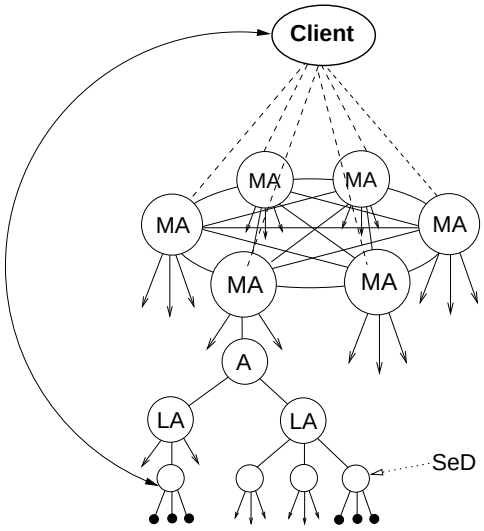

Figure 1: hierarchical organization of DIET.

on every node of the hierarchy to collect resource availabilities, which are then used by FAST.

DIET is also designed to take into account the data location during scheduling. Data are kept as long as possible on (or near to) the computational servers on which they have been computed to minimize transfer times. This kind of optimization is mandatory to obtain performance on a wide-area network.

Finally, NES environments like Ninf and NetSolve are implemented using a classic socket communication layer nevertheless several problems to this approach have been pointed out such as the lack of portability or the limitation of opened sockets. Distributed object environments, such as *Java*, *DCOM* or Corba have proven to be a good base for building applications that manage access to distributed services. They provide transparent communications in heterogeneous networks, but they also offer a framework for the large scale deployment of distributed applications. Moreover, Corba systems provide a remote method invocation facility with a high level of transparency. This transparency should not dramatically affect the performance, communication layers being well optimized in most Corba implementations [8]. Thus, Corba has been chosen as a communication layer in DIET.

In the next section, we design the hierarchical and distributed scheduler used in DIET. The scheduler is implemented using the well-known PIF scheme [6, 15]. In

the first subsection, we quickly recall how the PIF scheme works on a tree network and we describe how we use the PIF scheme to deal with requests submitted to the scheduler. Fault tolerance mechanisms are also discussed.

3 The Distributed Scheduling Algorithm

3.1 Description of the PIF Algorithm

The PIF [3, 6, 15] scheme works in two phases. The first phase is called the *broadcast* phase. The root of the tree network initiates the broadcast phase by broadcasting a message m to its descendants. Then, its descendants (except the leaves) participate in this phase by forwarding the message to their descendants. Basically, during the broadcast phase, m is propagated in the whole tree. Once the leaves are reached by the broadcast phase (i.e., m), since the leaves have no descendant, they notify the termination of the broadcast phase by sending a feedback message to their parent. This initiates the *feedback* phase. When the parent receives the feedback messages from all its descendants, it sends a feedback message to its own parent, and so on. So, eventually, the root receives a feedback message from all its descendants. This marks the end of the feedback phase. In other words, all nodes acknowledged the receipt of m to the root.

We now describe how the PIF scheme is used to proceed a request from a client to a DIET server.

3.2 Application of the Algorithm to DIET

Each client is assigned with a unique MA. The client sends the request to the corresponding MA which initiates a broadcast phase by propagating the request toward each available agent in its subtrees. So, each LA connected in the hierarchy eventually receives the client's request and the broadcast phase ends on the servers.

On each server, the FAST module included into the SeD computes the resources required to serve the request and the expected computation time. If a server is able to fulfill the request, FAST makes the reservation for the required resources. In any case, every server initiates the feedback phase by returning the forecast of the execution time to its parent LA.

According to the feedback phase synchronization, the LA waits for the FAST response from each of its descendant and chooses the identity of the most "appropriate" server (or list of servers for scheduling purposes), i.e., the fastest or the cheapest one(s). Next, following the feedback phase description, it sends the result to his

parent node and releases the servers which were not selected (using the PIF scheme in the corresponding subtree). Stage by stage, and following the same scheme, the MA eventually receives the selected server ID(s).

Finally, the MA sends the server ID(s) to the client which in turn contacts the server(s) to initialize the computation. Then, data are sent. The server(s) is now able to solve the requested problem. When the computation is done, the results are sent back to the client. All resources used to fulfill the request are released.

Two problems can occur with the algorithm previously presented. If every agent waits until all its son nodes have answered, and if some part of the hierarchy has crashed, we can have a infinite wait. Moreover with one mechanism of reservation of resource, DIET must process cross requests, i.e. requests sent by different clients for the same resource. In the next section, we explain how to deal with both problems.

3.3 Fault Tolerance Mechanisms

Processing of a Server Failure

In order to take into account servers failure we added a time-out at the LA level. The resulting algorithm for the server front-end (LA) is given in Algorithm 1.

The value of the timer α represents the time during which DIET waits for the first server's answer. If no server responds, the LA replies to its father that no server is available for the current request. If at least one server answers, a second timer equal to β is started. The purpose of this timer is to fix a compromise between the response time of the scheduler and the aggregation of the answers of different servers. Without this timer, one does not obtain the most effective server but the one which answered the most quickly to FAST. α and β depend on the number of servers. We can further optimize the response time by decreasing the timer β when the number of server responses increases.

Remark that the `Ireceive` call is an asynchronous function. If a server is not ready to answer one chooses the following according to a ring.

Processing of an Agent Failure

The failure of any agent in the hierarchy could also lead to an infinite wait and the lost of a branch of the scheduler tree. To avoid such wait, we set an other timer σ which depends of the depth of the tree (see Algorithm 2).

In this case, the `Ireceive` function can be seen like as synchronous. However, it can still be interrupted by the time-out σ.

Algorithm 1 LA algorithm using a time-out

```
timer = 0
time_out = α
current_server=0
response=0
while ((timer<time_out) OR (response! =nb_server)) do
  if Memory_response[current_server] != TRUE then
    if Ireceive(FAST_INFORMATION, S_current_server) == OK then
      timer = 0
      time_out = β
      response++
      Memory_response[current_server] = TRUE
    end if
  end if
  current_server++
  if current_server == nb_server+1 then
    current_server = 0
  end if
end while
```

Algorithm 2 Agent algorithm using a time-out

```
timer = 0
time_out = σ
for all leaf of the current agent do
  while (Ireceive(SCHEDULER_INFORMATION, Agent_leaf) == OK) OR
  (timer<time_out) do
    nothing
  end while
end for
```

Note that no Agent knows the numbers of server (in contrast to an LA). So, it seems difficult to design an algorithm using two time-out (as for the LA algorithm) because no way allows to compute the second time-out.

Concurrent Requests Processing

From the client point of view, the resource (server) allocated by the MA has to be reserved. Indeed, if two clients send two requests and the chosen server is the same (because this server is available during the FAST step), a conflict can occur. When the second client asks the server for the resolution of its request, the server can be already working on the first client's request.

To avoid this problem, we introduce a time-stamp mechanism in the FAST calls than make the resource reservation sequential. The drawback of this solution is the following. The capacities of the server can be underestimated when resources are released. We lose in precision for the forecast but gain of system stability for the client.

3.4 Experiments

The first simulation shows the impact of the hierarchy. For the sake of simplicity, we used an homogeneous network similar to FAST Ethernet and each server depends on the same CPU. Three kinds of DIET architectures have been defined to manage 64 servers (Table 1 describes the amount of different DIET components). The duration of each task is 100 seconds with no other load. Indeed, SIMGRID2 computes the running time using the CPU speed of the host (10 Mflops/s) and the amount of processing (in Mflop) needed to process the task on a server (1000 Mflop). In this simulation no reservation system is taken into account.

Hierarchical Level	**MA**	**Agent**	**LA**	**Server**
Level 1	1	2	4	16
Level 2	1	6	8	8
Level 3	1	14	16	4

Table 1: Amount of elements used to manage 64 servers. The column "Server" is the amount of servers depending on one LA.

Figure 2 shows the impact of the hierarchical model. Two factors explain this result. We do not have to prove that the broadcast on a tree is better than a straightforward round robin algorithm to find the best server. The second factor depends on the ratio between the time to send a request and the time to launch the computing task. Due to simultaneous requests send, a server gives the same response to a set of request. Thus several tasks choose the same server. When the

depth of the hierarchy increases, the response time increases also and the forecast error of resource availability decreases. The computing step begins after the end of the resource search step. Thus the forecasting system can take into account the result of computing more quickly. Moreover, this simulation shows the interest of reservation system previously described in Section 3.

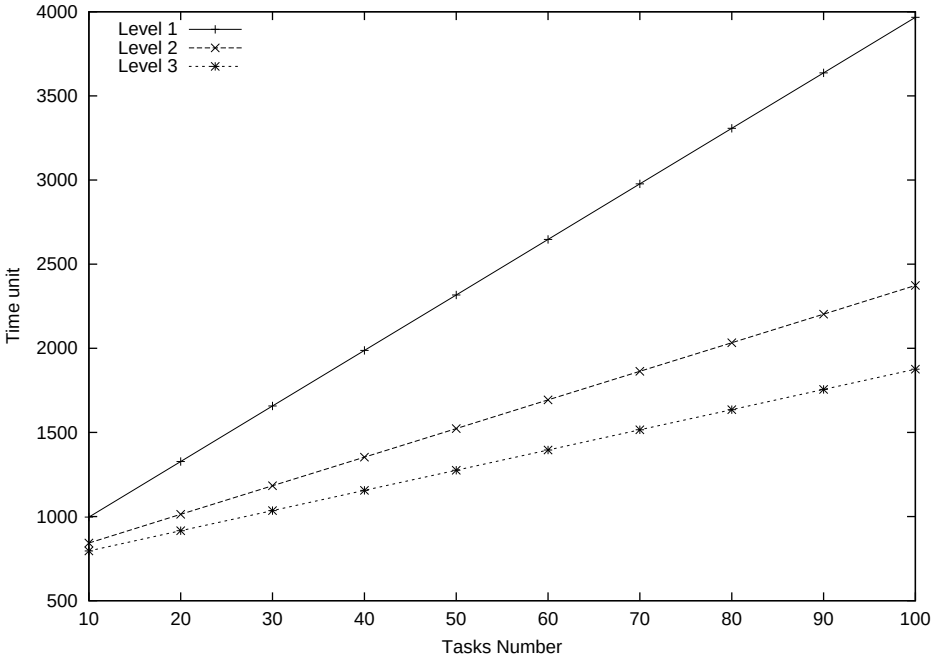

Figure 2: Evaluation of three hierarchical levels.

In a second experiment, we simulate DIET architecture with and without a reservation mechanism system. In this simulation, each task requires 10000 Mflops (1000 seconds in a stand-alone mode on a server). Figure 3 shows how it is important to include a reservation mechanism.

4 Conclusion and Future Work

In this paper, we have presented the application of the PIF algorithm to a Network Enabled Server environment. Hierarchical scheduling is applied to improve the

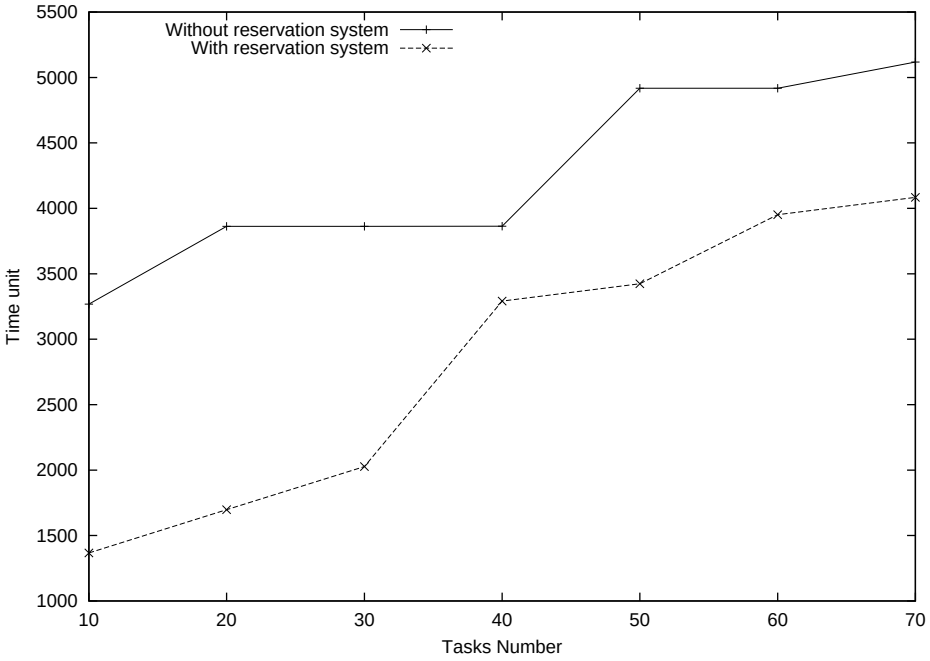

Figure 3: Evaluation of scheduler with reservation system and without in hierarchical platform (level 1).

scalability of the overall architecture and fault tolerance problems are addressed using timers. The simulation shows that gains can be obtained using a hierarchical platform.

Our future work consists in adding scheduling heuristics on each server to be able to manage tasks with data dependences. Moreover, as our environment allows to leave the data in place after the computation of one request, we need to add specific redistribution schemes between the servers. About fault tolerance, even if the presented approach avoids infinite waits, we need also to be able to restart agents or servers when a failure occurs. Finally, we are currently porting several applications on the DIET environment that will highlight specific problems due to their computation and data distribution needs.

References

[1] D. Andersen and T. McCune. H-SWEB: a Hierarchical Scheduling System for Distributed WWW Server Clusters. *Concurrency: Practice and Experience*, 12:189–210, 2000.

[2] D. Arnold, S. Agrawal, S. Blackford, J.J. Dongarra, M. Miller, K. Sagi, Z. Shi, and S. Vadhiyar. Users' Guide to NetSolve V1.4. Computer Science Dept. Technical Report CS-01-467, University of Tennessee, Knoxville, TN, July 2001. http://www.cs.utk.edu/netsolve/.

[3] A. Bui, A.K. Datta, F. Petit, and V. Vilain. Optimal PIF in Tree Networks. pages 1–16, 1999.

[4] E. Caron, F. Desprez, F. Lombard, J.-M. Nicod, M. Quinson, and F. Suter. A Scalable Approach to Network Enabled Servers. In *Proceedings of EuroPar 2002*, Paderborn, Germany, 2002.

[5] E. Caron and F. Suter. Parallel Extension of a Dynamic Performance Forecasting Tool. In *Proceedings of the International Symposium on Parallel and Distributed Computing*, pages 80–93, Iasi, Romania, Jul 2002.

[6] E.J.H. Chang. Echo algorithms: depth parallel operations on general graphs. *IEEE Transactions on Software Engineering*, SE-8:391–401, 1982.

[7] S.P. Dandamudi and S. Ayachi. Performance of Hierarchical Processor Scheduling in Shared-Memory Multiprocessors Systems. *IEEE Transactions on Computers*, 11(11):1202–1213, November 1999.

[8] A. Denis, C. Perez, and T. Priol. Towards high performance CORBA and MPI middlewares for grid computing. In Craig A. Lee, editor, *Proc. of the 2nd International Workshop on Grid Computing*, number 2242 in LNCS, pages 14–25, Denver, Colorado, USA, November 2001. Springer-Verlag.

[9] F. Desprez, M. Quinson, and F. Suter. Dynamic Performance Forecasting for Network Enabled Servers in a Metacomputing Environment. In *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2001)*. CSREA Press, June 2001.

[10] I. Foster and C. Kesselman (Eds.). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1998.

[11] S. Matsuoka, H. Nakada, M. Sato, and S. Sekiguchi. Design Issues of Network Enabled Server Systems for the Grid. http://www.eece.unm.edu/~dbader/grid/WhitePapers/satoshi.pdf, 2000. Grid Forum, Advanced Programming Models Working Group whitepaper.

[12] H. Nakada, M. Sato, and S. Sekiguchi. Design and Implementations of Ninf: towards a Global Computing Infrastructure. *Future Generation Computing Systems, Metacomputing Issue*, 15(5-6):649–658, 1999. http://ninf.apgrid.org/papers/papers.shtml.

[13] M. Quinson. Dynamic Performance Forecasting for Network-Enabled Servers in a Metacomputing Environment. In *International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS'02)*, April 2002.

[14] J. Santoso, G.D. Van Albada, and P.M.A. Sloot. Simulation of Hierarchical Resource Management for Meta-Computing Systems. *International Journal of Foundations of Computer Science*, 12(5):629–643, 2001.

[15] A. Segall. Distributed network protocols. *IEEE Transactions on Information Theory*, IT-29:23–35, 1983.

[16] A. Takefusa, H. Casanova, S. Matsuoka, and F. Berman. A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid. In *the 10th IEEE Symposium on High Performance and Distributed Computing (HPDC'01)*, San Francisco, California., August 2001.

[17] T. Thanalapati and S. Dandamudi. An Efficient Adaptive Scheduling Scheme for Distributed Memory Multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 12(07):758–768, July 2001.

[18] R. Wolski, N. T. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computing Systems, Metacomputing Issue*, 15(5–6):757–768, Oct. 1999.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399